Real-Time Multimedia System Analysis: A Breeze in Linux
Modern infotainment processors have sophisticated hardware accelerators for high-resolution video processing, but still there are some use-cases that stretch them to the limit, requiring near real-time processing in realizing them (e.g. BD). On general purpose operating systems (GPOS), meeting these requirements can be a challenge, resulting in video artefacts like jerkiness, freeze, audio-video sync loss, and distortion. This article will take you through a brief overview of typical media processing solution, issues faced in achieving real-time audio video rendering, and tools in Linux to analyze and fix such issues while taking the BD video/graphics data-path as an example.

Keywords—Kernel Tracers, trace-cmd, Blu-ray, de-interlace, blending

Abstract

Author: Chetan Sethi, System Software Architect, In-Vehicle Infotainment
# Table of Content

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>04</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
<tr>
<td>About The Author</td>
<td>11</td>
</tr>
<tr>
<td>About Sasken</td>
<td>12</td>
</tr>
</tbody>
</table>
Introduction

Typical media processing solutions like BD, STB, disc players need to process video and graphics data and render it. Different post processing operations like de-interlacing, alpha blending, color conversion, scaling, etc. are accomplished using variety of hardware accelerators to prepare one composite signal to be rendered.

Typical video/graphics sub-system post the decoding stage consists of the following:

- Post processing of video, image
- Blending videos and graphics
- Rendering video, image, graphics

BD composite signal consists of primary video, secondary video, and graphics. For interlaced video content, de-interlacing operation is performed and then primary video, secondary video, and BD graphics signals are blended. This composite BD signal is then finally blended with user graphics before rendering onto the LCD display.
In a general software architecture design for video/graphics composite signal rendering use cases, parallel execution of independent hardware accelerators like de-interlacer, Graphics Processing Unit (2S/3D Acceleration)/Blender and display sub-system are highly desirable to achieve optimal performance and throughput. The key idea is to reduce the interdependency, define independent features/task so that the execution is done in true parallel way to feed data at real-time intervals.

At the software level, a pipeline based threading model can be used for effective usage of these accelerators by defining one execution context (thread) per hardware accelerator. These threads are responsible picking up buffer for processing, executing post processing operation using the hardware blocks and outputting the resultant buffer to next hardware module. These execution contexts can have input and output queues defined so that, output of each stage can be fed into input of another stage.

Figure 2: Typical BD Software stack

Figure 3: Pipeline based threading model
The software stack for video/graphics sub-system needs to ensure that it is capable of rendering the frames at LCD refresh rate. Issues like jerky video, audio-video sync loss, distortion, video freeze, black screen can be noticed while achieving real-time rendering. Multiple reasons like frame drops, reception of delayed frames at rendering level, multiple user-kernel transactions, scheduling issues, locking contentions, clock related issues, high interrupt/DMA loads, etc. can result in these.

Finding the root cause of above mentioned artefacts, invariably requires an insight into what the system as a whole is doing at the instant of the artefact. Logs are of limited use when the analysis has to span multiple independently-developed components, system behavior, and long duration execution.

Kernel tracing in Linux provides a light-weight, time-accurate capability to inspect system events occurring between specific points in the application execution. This information is invaluable in ascertaining the system’s impact on the performance of a specific application and identifying the bottlenecks. In Linux these can be achieved via trace-cmd, tool that initiates the trace operation and generates the log and KernelShark, a GUI front-end tool for visual inspection of the log.

To ensure, the rendering is done flawless and in real-time, it’s essential that each block in video/graphics software pipeline does its unit task in well-defined real-time boundary. To start with ensuring this, Kernel traces can be used for accurate time measurement of different hardware IPs.
Kernel traces can be used for figuring out scheduling issues like higher priority thread of one particular process not allowing video rendering thread belonging to other process to schedule, leading to video breaks/jerky video issues. Such issues can be resolved by appropriately tuning the thread priorities of both the threads.
Kernel tracers have distinctive capability of common logging between user space and kernel space which can be extremely useful in figuring out issues like video freeze due to locking contentions at application layer.

Figure 6: User/Kernel Logging (Command: `#trace-cmd record –e ftrace`)

```
<....> 2469 [000] 122.055176: bprint: do_mlock: do_mlock 388
<....> 2469 [000] 122.055206: bprint: mlock_fixup: mlock_fixup 329
<....> 2469 [000] 122.055206: bprint: mlock_fixup: mlock_fixup 382
<....> 2469 [000] 122.055206: bprint: do_mlock: do_mlock 435
<....> 2469 [000] 122.055206: bprint: do_mlock_pages: do_mlock_pages 411
<....> 2469 [000] 122.055206: bprint: __mlock_vma_pages_range: __mlock_vma_pages_range 171
<....> 2469 [000] 122.055206: bprint: __mlock_vma_pages_range: __mlock_vma_pages_range 198
<....> 2469 [000] 122.055206: bprint: mlock_vma_page: mlock_vma_page 84
<....> 2469 [000] 122.055207: bprint: sys_mlock: sys_mlock 228
<....> 2469 [000] 122.055231: bprint: c00ac008 directshm_create_or_connect 127
<....> 2469 [000] 122.055267: bprint: c00ac008 directshm_create_or_connect 139
<....> 2469 [000] 122.055298: bprint: c00ac008 directshm_create_or_connect 135
<....> 2469 [000] 122.055298: bprint: c00ac008 directshm_create_or_connect 142
<....> 2469 [000] 122.055298: bprint: c00ac008 del_open 1012
<....> 2469 [000] 122.055329: bprint: c00ac008 del_open 1029
```
Kernel tracers provide interrupt statistics that can be captured using trace-cmd. Also trace-cmd extends interface for easy adaptation of data in excel format for further analysis. As can be seen in below example, one interrupt occurs per milliseconds indicating a very high rate of unexpected interruption adding to un-necessary load on the system. This particular instance of high rate of interrupt was traced to a power management issue in one of the video post processing hardware IPs.

Kernel tracers assist in analyzing system calls. As can be seen in below example, one particular system call (NR 91- sysmunmap) is taking around 235 milliseconds of time for returning. Such method is extremely useful in analyzing system specific issues like tuning of DMA channel priorities.

<table>
<thead>
<tr>
<th>TID</th>
<th>Time Stamp</th>
<th>System Call</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1944</td>
<td>1439.713837</td>
<td>sys_enter</td>
<td>NR 91 (95503000, 1c2000, 0, 0)</td>
</tr>
<tr>
<td>1944</td>
<td>1439.713857</td>
<td>irq_handler_entry</td>
<td>irq=30 name=arch_timer</td>
</tr>
<tr>
<td>1944</td>
<td>1439.713857</td>
<td>hrtimer_cancel</td>
<td>hrtimer=0xca6b3f28</td>
</tr>
<tr>
<td>1944</td>
<td>1439.713857</td>
<td>hrtimer_expire_entry</td>
<td>hrtimer=0xca6b3f28</td>
</tr>
<tr>
<td>1944</td>
<td>1439.713857</td>
<td>sched_stat_sleep</td>
<td>comm=trace-cmd pd=17003</td>
</tr>
<tr>
<td>1944</td>
<td>1439.79306</td>
<td>sched_wakeup</td>
<td>1944?? + 1946100?</td>
</tr>
<tr>
<td>1944</td>
<td>1439.79306</td>
<td>hrtimer_expire_exit</td>
<td>hrtimer=0xe9fadac8</td>
</tr>
<tr>
<td>1944</td>
<td>1439.79306</td>
<td>irq_handler_exit</td>
<td>irq=30 ret=handled</td>
</tr>
<tr>
<td>1944</td>
<td>1439.75892</td>
<td>softirq_entry</td>
<td>vec=1 [action=TIMER]</td>
</tr>
<tr>
<td>1944</td>
<td>1439.75895</td>
<td>softirq_exit</td>
<td>vec=1 [action=TIMER]</td>
</tr>
<tr>
<td>1944</td>
<td>1439.806702</td>
<td>irq_handler_entry</td>
<td>irq=30 name=arch_timer</td>
</tr>
<tr>
<td>1944</td>
<td>1439.806732</td>
<td>hrtimer_expire_entry</td>
<td>hrtimer=0xc1211998</td>
</tr>
<tr>
<td>1944</td>
<td>1439.806732</td>
<td>softirq_raise</td>
<td>vec=1 [action=TIMER]</td>
</tr>
<tr>
<td>1944</td>
<td>1439.806732</td>
<td>rcu_utilization</td>
<td>078c544</td>
</tr>
<tr>
<td>1944</td>
<td>1439.849371</td>
<td>kfree</td>
<td>(drm_gem_vm_close+0x28)</td>
</tr>
<tr>
<td>1944</td>
<td>1439.849371</td>
<td>kmem_cache_free</td>
<td>(remove_wma+0x48)</td>
</tr>
<tr>
<td>1944</td>
<td>1439.849371</td>
<td>sys_exit</td>
<td>NR 91 = 0</td>
</tr>
<tr>
<td>1944</td>
<td>1439.713837</td>
<td>sys_enter</td>
<td>NR 91 (95503000, 1c2000, 0, 0)</td>
</tr>
</tbody>
</table>

Figure 7 Interrupt statistics (#trace-cmd record –e irq)  
Figure 8 System Call statistics (#trace-cmd record –e all)
Kernel-shark and trace-cmd are ideal tools for analyzing performance related issues/latencies and they have quite a few advantages for performance measurement over traditional methods. Apart from performance measurement quite a few advance debugging techniques are exposed by them.

The key advantages of the trace-cmd and KernelShark tools are:

- It can be used for debugging or analyzing latencies and performance issues.
- It has minimal impact on the CPU load and is very accurate.
- It has ability to trace syscall entry and exit, and signal delivery, to a process (which can also used for debugging a process)
- Excellent user level representation tools like kernel shark.
- Dynamic control for recording/stopping traces.
- Complete thread level scheduling statistics for the entire system can be generated.
- Provide interface for plotting graphs in excel sheet.
- It gives detailed information on multiple system parameters like interrupts, kernel events etc.

Gives unified interface for logging across kernel and user space with single log file generation.

Conclusion
About The Author

Chetan Sethi has rich experience in Automotive In-Vehicle Infotainment product development, Linux, Multimedia, and the Consumer Electronics space. He has technical contribution in multiple areas spanning from performance analysis of complex systems, base porting in Linux, audio DSP algorithm development to name a few. Also he has immense exposure of working with Japanese Tier-1s for entire product development cycle without compromising upon stringent quality expectations and strict deadlines. At Sasken, has the responsibility of delivering automotive In-Vehicle Infotainment sub-system for future models to a Japanese Tier-1.
About Sasken

Sasken is a specialist in Product Engineering and Digital Transformation providing concept-to-market, chip-to-cognition R&D services to global leaders in Semiconductor, Automotive, Industrials, Smart Devices & Wearables, Enterprise Grade Devices, Satcom, and Retail industries. With over 27 years in Product Engineering and Digital Transformation and 70 patents, Sasken has transformed the businesses of over a 100 Fortune 500 companies, powering over a billion devices through its services and IP.
Real-Time Multimedia System Analysis: A Breeze in Linux